

APPLICATION FOR U.S. PATENT

APPARATUS AND METHODS FOR INTEGRATING APIs FOR
PROGRAM EXECUTION MANAGEMENT

INVENTOR: Hideya Kawahara
520 Walker Dr., Apt. 35
Mountain View, California 94043
Citizen of Japan

ASSIGNEE: SUN MICROSYSTEMS, INC.
901 San Antonio Road
Palo Alto, CA 94303

ENTITY: LARGE

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
(510) 843-6200

APPARATUS AND METHODS FOR INTEGRATING APIS FOR PROGRAM EXECUTION MANAGEMENT

By Inventor: Hideya Kawahara

5 BACKGROUND OF THE INVENTION

The invention relates generally to software applications and, more particularly, to methods and apparatus for controlling execution of a software component that resides within an isolated execution unit.

In many software applications, there is a need for isolated execution units. An isolated execution unit generally does not share its states with other software modules which are not a part of the isolated execution unit. Conventionally, a software module that does not reside within an isolated execution unit cannot communicate with other software modules that do reside in the isolated execution unit. For example, an isolated execution unit does not share memory with another software module that does not reside within the isolated execution unit. Since isolated execution units do not share states, a particular isolated execution unit is protected from an erroneous behavior running in another isolated execution unit. Examples of isolated execution units include a "process" in a Unix operating system or a Logical Virtual Machine on a Java platform.

Each isolated execution unit may also create one or more internal software components. For example, a isolated execution unit may take the form of a browser application that spawns a software component in the form of an applet. Several different software component models have been proposed, including Java applet, Java Xlet and Java servlet. Those software component models typically have unique APIs

(Application Programming Interfaces) that are different from each other.

There are some cases where it is better to locate and execute these software components in different isolated execution units than the one that creates them in order to isolate erroneous behavior. For example, a web browser spawns an isolated execution unit; it instantiates a new applet in it; then the web browser starts the execution of the applet. However, there are several difficulties to practicing this scheme. Figure 1 is a diagrammatic representation of a typical isolated execution unit setup. As shown, a isolated execution unit 102 includes an internal software component 104. The isolated execution unit has API 108, and the software component 104 has API 110 that may differ from the isolated execution unit's API 108. A parent program 106 communicates with the isolated execution unit 102 through the API 108 of the isolated execution unit 102. However, the parent program 106 cannot communicate with the software component 104 since the software component's API 110 is not accessible by the parent program 106. That is, the software component's API 110 is "isolated" from the parent program 106.

Since the parent program 106 cannot communicate with the software component, the parent program 106 cannot access API 110 of the software component 104. The parent program may need to access the API 110 to control the software component for any number of reasons. In one example, the isolated execution unit 102 represents a web browser and the software component 104 is an applet. If the applet 104 misbehaves, it is desirable to shut down the applet. However, in this setup the applet can only be terminated by shutting down the entire browser 102. That is, the browser 102 is needlessly shut down. Of course, shutting down the browser is typically very undesirable. In another example, the parent

program cannot initialize the state of the isolated execution unit before execution of the software component to a desired state. For example, this initialization may include assigning initial values to execution environment variables.

A conventional technique for solving this problem is to provide multiple APIs within the isolated execution unit for each kind of software component within the isolated execution unit. Figure 2 is a diagrammatic representation of a communication mechanism, where an isolated execution unit 202 has been modified to expose an internal component's API. As shown, the isolated execution unit 202 includes API 208 for software component 204. A parent program 206 may then communicate with software component 204 through the exposed API 208.

This approach has several associated disadvantages. Since an API must be created for each software component, a significant number of API's may have to be created. Additionally, modification of an isolated execution unit is not desirable. One reason is that modification of the isolated execution unit can be rather complex and time consuming. Also, modification of the isolated execution unit for each software component cancels out the advantages of using software components in the first place, *e.g.*, the ability to simply use software components "as is" without modification.

Accordingly, there is a need for improved mechanisms for controlling or monitoring execution of a software component that is associated with an isolated execution unit.

SUMMARY OF THE INVENTION

Broadly speaking, the present invention fills these needs by providing apparatus and methods for controlling or monitoring execution of a target software component within an isolated execution unit. In general terms, an intermediary software component is introduced within the isolated execution unit. In order to communicate back to the parent program, an inter isolation communication mechanism may be used. In that case, the intermediary software component may be designed or modified to perform initialization of such a communication mechanism and use the communication mechanism to communicate with the parent. This intermediary component program can initialize the isolated execution unit, and then start a target software component within the isolated execution unit. That is, the intermediary component establishes a communication path back to the parent (e.g., using an inter isolation communication). The intermediary component may then communicate with the target software component using the target component's unchanged API and mediates the communication back to the parent using the established communication path.

In one embodiment, a method for controlling or monitoring a target software component of an isolated execution unit is disclosed. An intermediary software component is introduced within an isolated execution unit. An identifier of a target software component is indicated to the intermediary software component. The target software component having the indicated identifier is initiated within the isolated execution unit. A communication path is established between the intermediary software component and an external program that is outside of the isolated execution unit whereby the external program can control or monitor the target software

component via the established communication path.

In a further implementation, the identifier of the target software component is provided by the external program. Preferably, the established communication path uses an inter isolation communication protocol. For example, the inter isolation communication protocol is a remote method invocation technique. Additionally, the communication path is established by the intermediary software component. In one implementation, the isolated execution unit is initialized by the intermediary software component. In a further implementation, one or more parameters for initializing the isolated execution unit may be indicated by the external program to the intermediary software component. The initialization of the isolated execution unit is based on the indicated one or more parameters.

In a further aspect, an execution control parameter is indicated to the intermediary software component and the indicated execution control parameter is invoked on the target software component using an application programming interface (API) of the target software component. Additionally, the execution control parameter is a request that has a first format of an inter isolation communication protocol. The intermediary software component translates the first format into a second format that is acceptable by the API of the target software component.

In another implementation, the intermediary software component receives a result from the target component in response to the invoked execution control parameter and sends the result to the external program. The result has a first format that is acceptable by the API of the target software component. The intermediary software component translates the first format into a second format that is an inter

isolation communication protocol before sending the result to the external program.

In another aspect, the invention pertains to a computer implemented system operable to control or monitor a target software component of an isolated execution unit. The computer system includes an isolated execution unit, an intermediary software component within the isolated execution unit, and an external program that is outside of the isolated execution unit. The external program is configured to indicate an identifier of a target software component to the intermediary software component. The intermediary software component is configured to start the target software component having the indicated identifier within the isolated execution unit and establish a communication path between the intermediary software component and the external program whereby the external program can control the target software component via the established communication path. The components of the computer implemented system are further adapted to provide one or more of the above described methods. In another embodiment, the invention pertains to a computer readable medium having instructions for controlling or monitoring a target software component of an isolated execution unit. The computer readable medium includes computer code for performing the above described methods.

Advantages of the present invention will become apparent upon reading the following detailed descriptions and studying the various figures of the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings.

5 Figure 1 is a diagrammatic representation of a typical isolated execution unit setup.

Figure 2 is a diagrammatic representation of a communication mechanism, where an isolated execution unit has been modified to expose an internal component's API.

10 Figure 3 is a diagrammatic representation of a communication mechanism for controlling execution of a target software component of an isolated execution unit in accordance with one embodiment of the present invention.

Figure 4 is a flowchart illustrating a procedure for communicating with a target component of an isolated execution unit in accordance with one embodiment of
15 the present invention.

Figure 5 is a flowchart illustrating the operation of Figure 4 of communicating between the parent program and the target component in accordance with one embodiment of the present invention.

Figure 6 is a diagrammatic representation of a general-purpose computer
20 system suitable for implementing the present invention.

Figure 7 is a diagrammatic representation of a virtual machine which is supported by computer system of Figure 6, and is suitable for implementing the

DETAILED DESCRIPTION OF THE EMBODIMENTS

Reference will now be made in detail to specific embodiments of the invention. While the invention will be described in conjunction with specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention.

Figure 3 is a diagrammatic representation of a communication mechanism 300 for controlling execution of a target software component of an isolated execution unit in accordance with one embodiment of the present invention. In general terms, an external program 302 sets up an intermediary component 306 within an isolated execution unit 304 (e.g., the intermediary component becomes a part of the isolated execution unit). In the illustrated embodiment, the external program is a parent program of the isolated execution unit. However, the execution unit may have been created by another external program or parent program. The intermediary component 306 then sets up a communication path 314 to the external program 302. The external program 302 may then make a request through path 314 to the intermediary component 306. The intermediary component 306 then translates the request and sends it to a target component 308 through communication path 316 and the target component's API 318. Results in response to the request may also be sent from the

target component 308, translated by the intermediary component 306, and sent to the external program 302 through communication path 316. In sum, the intermediary component 306 communicates with the external program via the established path 314 and communicates with the target component 308 through the target's own API 318.

5 Figure 4 is a flowchart illustrating a procedure 400 for communicating with a target component of an isolated execution unit in accordance with one embodiment of the present invention. Of course, the procedure of Figure 4 may be applied to multiple components within a same or different isolated execution unit. Initially, an external program may create a new isolated execution unit in operation 402.
10 Alternatively, the isolated execution unit may already have been created by another external program. The external program then starts an intermediary component within the new isolated execution unit that is given a target components name and one or more desired state(s) for the isolated execution unit and/or target component in operation 404.

15 The intermediary component then starts execution in operation 406. The intermediary component then initializes the isolated execution unit into the one or more desired state(s) supplied by the external program in operation 408. The intermediary component also starts the name target component in operation 410. The target component may also be initialized into one more desired states previously
20 supplied by the external program. The states may include any configurable state. For example, a particular variable may be set, which may include setting the current working directory, the maximum number of file descriptors that the isolated execution unit can use, a name of input data file, etc.

Additionally, the intermediary component establishes a communication path to the external program in operation 412. The external program may then communicate with the intermediary component via the established communication path so as to communicate with the target component in operation 414. This communication may be in any suitable form for controlling or monitoring execution of the target component. For example, the external program may make a method request to the target component via the established communication path.

Figure 5 is a flowchart illustrating the operation 414 of Figure 4 of communicating between the external program and the target component in accordance with one embodiment of the present invention. The illustrated operation 414 of Figure 5 merely represents an example communication sequence between an external program and a target component of an isolated execution unit. Of course, there are numerous forms that such a communication could take. In the illustrated embodiment, the external program initiates a control parameter or request to be sent to the target component in operation 502. The request is in a form for communicating through the established communication path. Preferably, the communication path uses an inter isolation communication protocol. In a specific implementation, the inter isolation communication protocol is a remote method invocation (RMI) technique. By way of examples, the request may take the form of a command for controlling execution of the target component (*e.g.*, a method request) or a request for information from the target component to monitor execution of the target component. Other examples include requests to change the presentation of the GUI (graphical user interface), to perform calculation of data, send and receive information to and from server machines, etc.

program via the established communication path in operation 512. That is, the results may be translated from a format suitable for the target component's API (e.g., applet API) into a format suitable for the established communication path (e.g., RMI).

Figure 6 illustrates a typical, general-purpose computer system suitable for implementing the present invention. The computer system 1030 includes any number of processors 1032 (also referred to as central processing units, or CPUs) that are coupled to memory devices including primary storage devices 1034 (typically a read only memory, or ROM) and primary storage devices 1036 (typically a random access memory, or RAM). The computer system may take any suitable form. For example, the computer system may be integrated with a navigational system or television set top box.

Computer system 1030 or, more specifically, CPUs 1032, may be arranged to support a virtual machine, as will be appreciated by those skilled in the art. One example of a virtual machine that is supported on computer system 1030 will be described below with reference to Figure 7. As is well known in the art, ROM acts to transfer data and instructions uni-directionally to the CPUs 1032, while RAM is used typically to transfer data and instructions in a bi-directional manner. CPUs 1032 may generally include any number of processors. Both primary storage devices 1034, 1036 may include any suitable computer-readable media. A secondary storage medium 1038, which is typically a mass memory device, is also coupled bi-directionally to CPUs 1032 and provides additional data storage capacity. The mass memory device 1038 is a computer-readable medium that may be used to store programs including computer code, data, and the like. Typically, mass memory device 1038 is a storage medium such as a hard disk or a tape which generally slower than primary storage

devices 1034, 1036. Mass memory storage device 938 may take the form of a magnetic or paper tape reader or some other well-known device. It will be appreciated that the information retained within the mass memory device 1038, may, in appropriate cases, be incorporated in standard fashion as part of RAM 1036 as
5 virtual memory. A specific primary storage device 1034 such as a CD-ROM may also pass data uni-directionally to the CPUs 1032.

CPUs 1032 are also coupled to one or more input/output devices 1040 that may include, but are not limited to, devices such as video monitors, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic
10 or paper tape readers, tablets, styluses, voice or handwriting recognizers, or other well-known input devices such as, of course, other computers. Finally, CPUs 1032 optionally may be coupled to a computer or telecommunications network, e.g., an Internet network or an intranet network, using a network connection as shown generally at 1012. With such a network connection, it is contemplated that the CPUs
15 1032 might receive information from the network, or might output information to the network in the course of performing the above-described mechanisms for managing target components. Such information, which is often represented as a sequence of instructions to be executed using CPUs 1032, may be received from and outputted to the network, for example, in the form of a computer data signal embodied in a carrier
20 wave. The above-described devices and materials will be familiar to those of skill in the computer hardware and software arts.

As previously mentioned, a virtual machine may execute on computer system 1030. Figure 7 is a diagrammatic representation of a virtual machine which is supported by computer system 1030 of Figure 6, and is suitable for implementing the

present invention. When a computer program, *e.g.*, a computer program written in the Java™ programming language, is executed, source code 1110 is provided to a compiler 1120 within compile-time environment 1105. Compiler 1120 translates source code 1110 into byte codes 1130. In general, source code 1110 is translated into
5 byte codes 1130 at the time source code 1110 is created by a software developer.

Byte codes 1130 may generally be reproduced, downloaded, or otherwise distributed through a network, *e.g.*, network 1012 of Figure 6, or stored on a storage device such as primary storage 1034 of Figure 6. In the described embodiment, byte codes 1130 are platform independent. That is, byte codes 1130 may be executed on
10 substantially any computer system that is running on a suitable virtual machine 1140.

Byte codes 1130 are provided to a runtime environment 1135 which includes virtual machine 1140. Runtime environment 1135 may generally be executed using a processor or processors such as CPUs 1032 of Figure 6. Virtual machine 1140 includes a compiler 1142, an interpreter 1144, and a runtime system 1146. Byte codes
15 1130 may be provided either to compiler 1142 or interpreter 1144.

When byte codes 1130 are provided to compiler 1142, methods contained in byte codes 1130 are compiled into machine instructions. In one embodiment, compiler 1142 is a just-in-time compile which delays the compilation of methods contained in byte codes 1130 until the methods are about to be executed. When byte
20 codes 1130 are provided to interpreter 1144, byte codes 1130 are read into interpreter 1144 one byte code at a time. Interpreter 1144 then performs the operation defined by each byte code as each byte code is read into interpreter 1144. That is, interpreter 1144 “interprets” byte codes 1130, as will be appreciated by those skilled in the art.

In general, interpreter 1144 processes byte codes 1130 and performs operations associated with byte codes 1130 substantially continuously.

When a method is invoked by another method, or is invoked from runtime environment 1135, if the method is interpreted, runtime system 1146 may obtain the method from runtime environment 1135 in the form of a sequence of byte codes 1130, which may be directly executed by interpreter 1144. If, on the other hand, the method which is invoked is a compiled method which has not been compiled, runtime system 1146 also obtains the method from runtime environment 1135 in the form of a sequence of byte codes 1130, then may go on to activate compiler 1142. Compiler 1142 then generates machine instructions from byte codes 1130, and the resulting machine-language instructions may be executed directly by CPUs 1032. In general, the machine-language instructions are discarded when virtual machine 1140 terminates. The operation of virtual machines or, more particularly, Java™ virtual machines, is described in more detail in The Java™ Virtual Machine Specification by Tim Lindholm and Frank Yellin (ISBN 0-201-63452-X) which is incorporated herein by reference.

Although only a few embodiments of the present invention have been described, it should be understood that the present invention may be embodied in many other specific forms without departing from the spirit or the scope of the present invention. By way of example, the operations involved with managing target components may be reordered. Steps may also be removed or added without departing from the spirit or the scope of the present invention.

Although the techniques for managing target components in accordance with

the present invention are particularly suitable for implementation with respect to a Java™ based environment, the techniques may generally be applied in any suitable object-based environment. In particular, the techniques are suitable for use in platform-independent object-based environments. It should be appreciated that the
5 methods may also be implemented in some distributed object-oriented systems. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.